Leistungsnachweis

Distributed and Mobile Systems

Autoren

Fidan Arsen, BWI-A-19

Dozent

Rothlin Walter

Modul

BWI-A19-5 Distributed and Mobile Systems

HWZ Hochschule für Wirtschaft Zürich

Lagerstrasse 5

8021 Zürich

Wallisellen, 23.11.2021

Inhaltsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1 Auftrag

Fach	Distributed & Mobile-Systems	**Bearbeitungszeit**	ca 7h
Titel	Weather Klasse		
Studiengruppe	BWI-A19	**Anzahl Aufgaben**	Einzelarbeit
Abgabe-Datum	23.11.21 / 17:00	**Bewertung**	Geprüft wird der Code nach vorgegebenen funktionalen und Qualitäts-Kriterien. Anhand eines Fachgespräches wird überprüft, ob der Student den Code und die darunter liegenden theoretischen Grundlagen verstanden hat. (Gewicht: 100%)

Erlaubte Hilfsmittel	• Alles inkl. Google, PyCharm, … **ausser 1:1 Abschreiben oder Kopieren** ohne Deklaration als Kommentar im Code.
Auftrag	• Design und Implementation einer allgemeinen Wetterstations-Klasse • Entwickeln einer CLI Test-Applikation, welche diese Klasse nutzt und zum Testen der Klasse verwendet werden kann.

Tabelle 1 Auftragsbeschreibung

1.2 Grobbeschreibung

Um einen Web-Service in Python zu nutzen, wird oft vom Service-Provider eine Klasse (Module) zur Verfügung gestellt. Diese Klasse kann der Entwickler verwenden, um Objekte zu kreieren und in der seine eigene Applikation nach OO-Richtlinien zu designen und zu implementieren.
Entwickeln Sie eine allgemeine WeatherStation-Class, welche für verschiedenste Web-Services für Wetterdienste verwendet werden kann. Ein konkreter Wetterdienst (z.B. OpenWeather) muss implementiert sein. Dabei ist das API des Wetterdienste nur in der Implementation sichtbar und somit für den Verwender der Klasse (Applikation) nicht.
Für diese Klasse entwickeln Sie eine CLI-Applikation, mit welcher die Klasse getestet werden kann.
Für diese Aufgabenstellung „packen“ Sie alles in ein einziges PythonFile und fügen Kommentare und Docstrings mit Ihren Überlegungen dazu.

1.3 Detailbeschreibung

Functional and Quality Requirements

 Lauffähige Test A pplikation rechtzeitig abgegeben

 Alles in einem File (ausnahmsweise für diese

 Filename: V orname_N achname_A19_DS.py (z.B. Rea_Vogel_A19 _DS.

 Cleancode Regeln berücksichtigt

 Wetter Klasse vorhanden

o Design d okumentiert (z.B. mit Class Diagramm)Diagramm), publiziert und in Klasse verlinkt

o Design stimmt mit Implementation überein

o __str__ and __eq__ implemented

o Initializer overloaded mit D efault V alues

o K lares , verständliches und einfaches Interf a c e

o Sichtbarkeit sregeln beachtet

o OO Design und Encapsulation nachvollziehbar

o Methoden Argumen te haben sinnvolle Default Werte

o Methoden haben allgemein strukturierte Return Values

o K lasse greift auf Wetterdienst Daten zu

 A pplication

o Eigene AppId gelöst

o User I nteraktivität vorhanden

 Test

o Test Fälle dokumentiert

o Test Statistik vorhanden

o Test Abdeckung genügend

o Positive wie negative Testfälle implementiert

o Test Driven approach erke nnbar

2. Kurzbericht

Das Ziel der WeatherStation Klasse ist eine benutzerfreundliche Bedienung aussenstehender Nutzer.

Somit wurden objektunabhängige Hilfs- Methoden/Funktionen als staticmethods deklariert.

Die Logik und das (Error) Handling der Verwendung der Parameter wird in den staticmethods bearbeitet, sodass der Benutzer jeweils nur das Objekt zu instanziieren und mit einer einzigen Methode das Objekt zu bearbeiten hat.

Es beinhaltet für die Auswahl der Stadt sowie anzuzeigenden Wetterdaten Interaktivität in den unterliegenden Methoden, die automatisch ausgeführt werden.

Ebenso werden die zu anzeigende Stadt und Wetterattribute auf Korrektheit überprüft, indem ein vordefiniertes Set von Wetterattributen (gegeben von Service Provider) und eine Liste verfügbarer Städte (Download vom Service Provider im Runtime) als GoldenSource verwendet werden.

Eingaben können case-insensitive gemacht werden.

2.1 Testing

Beim Testen wurden die möglichen Kombinationen der Parameter durchgeführt und dokumentiert.

1. Das Objekt mit definierten Parametern instanziieren
   - *Demo = WeatherStation(access-*Key="3836093dde650898eb014e6f27304646",unit="imperial",lang="de")
2. Das Objekt ohne definierte Parameter instanziieren
   - *Demo = WeatherStation()*
3. Die Methode mit definierten Attributen jedoch ohne definierte Stadt ausführen
   - *Demo.GetWeather(customfield=["time", "city", "temp","SunriSe","DE*SCRIPTION"])
4. Die Methode mit definierten Attributen mit definierter Stadt ausführen
   - *Demo.GetWeather(city="belgrade", customfield=["time", "city", "temp"])*
5. Die Methode mit ohne Attribute und ohne definierte Stadt ausführen
   - *Demo.GetWeather()*
6. Die Methode ohne definierte Attribute jedoch mit definierter Stadt ausführen
   - *Demo.GetWeather(city="berlin")*
7. Die Methode mit falschen definierten Attributen ausführen
   - *Demo.GetWeather(city="berliin")*
8. Die Methode mit falsch definierter Stadt ausführen
   - *Demo.GetWeather(city="berlin", customfield=["tiime","city"])*

Alle Szenarien wurden mit erwartetem Ergebnis ausgeführt.

3. Quellenverzeichnis

Alle Zeilen wurden anhand eigenen Gedankengutes erstellt und geschrieben. Als Referenz dienten Beispiele von https://www.w3schools.com/ und dem Kursmaterial «Distributed and Mobile Systems»

4. Anhang

4.1 Quellcode

```
# ----------------------------------------------------------------------
# Name: Arsen_Fidan_A19_DS.py
#
# Description: Weather Station class for retrieving weather data over ex-
ternal Web Api Service including
# a cli test application for testing and demonstration purposes.
#
# Author: Arsen Fidan
#
# Date: 2021/11/23
#
# Version: 1.0
# Documentation: BWI-A19_LNW_DMS_ArsenFidan.pdf
# ----------------------------------------------------------------------

import requests
import json
import gzip
from datetime import datetime


class WeatherStation:

    def __init__(self, accessKey="4e4c04c2b96c9d94c2a5863a7cb41ab8",
unit="metric", lang="en"):
        self.accessKey = accessKey
        self.unit = unit
        self.lang = lang

    def __str__(self):
        return f"AccessID is: {self.accessKey} \nLanguage is: {self.lang}
\nUnits shown in: {self.unit}"

    def __eq__(self, other):
        if other.accessKey == self.accessKey:
            return True
        return False


    def GetWeather(self, city=None, customfield=None, console: bool =
True):
        """
        Only This Method must be used to retrieve Weather Data. Source is
OpenWeatherMag.Org
        All Parameter are optional and can be left empty. Handling are
inherited in static methods.

        -city(str):     defines City to retrieve weather data for
        -customfield[]: defines set of weather attribute to show
        -console:       defines if output wil be shown on console
        """

        print("\n ")
        print(self)
        city = self.ValidateCity(city)
        url = self.CreateURL(city=city, unit=self.unit, lang=self.lang,
accessKey=self.accessKey)
```

```
        restData = self.GetRESTData(url)
        fields = self.ChooseFields(customfield)
        getWeather = self.AttributeMapping(rawObj=restData, cus-
tom=fields)

        output = getWeather

        if console:
            print("\n")
            for attribute, value in output.items():
                print(f"{attribute}: {value}")
        return output

    # ------------------- Object independent static functions -----------
--------


    @staticmethod
    def CreateURL(city, unit, lang, accessKey):
        """
        Creates URL customized for API of OpeanWeather. Parameters argu-
ments will be used from object.

        -city:       defines City to retrieve weather data for
        -unit:       defines set of weather attribute to show
        -accessKey:  defines accessKey for API access
        """

        url = f"https://api.openweather-
map.org/data/2.5/weather?q={city}&lang={lang}&units={unit}&appid={access-
Key}"
        output = url
        return output

    @staticmethod
    def GetRESTData(url):
        """
        Generic Method to use RestApi with GET, retrieve data and return
as json.
        """
        restData = requests.get(url)
        jsonData = json.loads(restData.text)
        output = jsonData
        return output

    @staticmethod
    def AttributeMapping(rawObj, custom=None):
        """
        This Method is used to whether use all existing weather attrib-
utes of OpenWeather or scale to predefined
        set of attributes provided by output of ChooseFields().
        Input can be empty.
        Raw data must be from OpenWeather.
        """
        weatherObj = {}
        attributes = {
            "Time": WeatherStation.ConvertUnixTime(rawObj["dt"]),
            "City": rawObj["name"],
            "Description": rawObj["weather"][0]["description"],
            "Sky": rawObj["clouds"]["all"],
            "Temp": rawObj["main"]["temp"],
            "TempMax": rawObj["main"]["temp_max"],
            "TempMin": rawObj["main"]["temp_min"],
```

```
            "Wind": rawObj["wind"]["speed"],
            "Sunrise": WeatherStation.ConvertUnixTime(rawObj["sys"]["sun-
rise"]),
            "Sunset": WeatherStation.ConvertUnixTime(rawObj["sys"]["sun-
set"])
        }
        if custom is not None:
            for attr, value in attributes.items():
                if attr.upper() in custom:
                    weatherObj.update({attr: value})
        elif custom is None:
            weatherObj = attributes

        output = weatherObj
        return output

    @staticmethod
    def ChooseFields(fields: list = None):
        """
        Method is validating specific list of custom fields against
available fields. This can be used generic.
        If input is wrong or an item not existing of predefined available
fields an error will warn you.
        Input can be case-insensitive.
        Input can be empty.
        -Fields: must be a list
        """
        customFields = []
        availableFields = ["Time", "City", "Description", "Sky", "Temp",
"TempMax", "TempMin", "Wind", "Sunrise",
                           "Sunset"]
        for item in range(len(availableFields)):
            availableFields[item] = availableFields[item].upper()

        if fields is not None:
            for attribute in fields:
                if attribute.upper() in availableFields:
                    customFields.append(attribute.upper())
                else:
                    print(f"Attribute {attribute} not existing, please
try again. \n")
                    exit()
            print("Attributes are: " + ', '.join(customFields))
        elif fields is None:
            customFields = None
            print("Attributes are: " + ', '.join(availableFields))

        output = customFields
        return output

    @staticmethod
    def GetCities():
        """
        Downloads all available cities for data of OpenWeather and gives
output as an array of cities.
        """
        cities = []
        url = "https://bulk.openweathermap.org/sample/city.list.json.gz"
        req = requests.get(url)
        obj = gzip.decompress(req.content)
        jsonData = json.loads(obj)

        for key in jsonData:
```

```
            cities.append(key["name"])
        output = cities
        return output

    @staticmethod
    def ValidateCity(city=None):
        """
        Validates inserted city against the downloaded cities from
GetCities Method.
        If city is empty an input will ask for feed.
        If city is wrong an output will warn and stop further processing.
        Cities can be used case-insensitive.
        """
        output = None
        cities = WeatherStation.GetCities()

        for item in range(len(cities)):
            cities[item] = cities[item].upper()

        if city is None:
            response = input("Enter city name: ")
            response = response.upper()
            if response not in cities:
                print(f"City NOT found: {response}")
                exit()
            else:
                print(f"City found: {response}.")
                output = response
        else:
            city = city.upper()
            if city not in cities:
                print(f"City NOT found: {city}")
                exit()
            else:
                print(f"City found: {city}")
                output = city

        return output

    @staticmethod
    def ConvertUnixTime(inputTime):
        """
        Simple timestamp converter to create readable time data
        """
        time = datetime.fromtimestamp(inputTime)
        output = str(time)
        return output


# ------------------- Object independent static functions ---------------
----


# ------------------------ Demo Cli App ---------------------------

"""
See documentation for explanation of instance Demos.
"""
# Demo = WeatherStation(access-
Key="3836093dde650898eb014e6f27304646",unit="imperial",lang="de")
Demo = WeatherStation()
```

```
"""
See documentation for explanation of Method Demos.
"""
#Demo.GetWeather(customfield=["time", "city", "temp","SunriSe","DESCRIP-
TION"])
#Demo.GetWeather(city="belgrade", customfield=["time", "city", "temp"])
Demo.GetWeather()
# Demo.GetWeather(city="berliin")
# Demo.GetWeather(city="berlin", customfield=["tiime","city"])



# ------------------------- Demo Cli App ---------------------------
```